

## FILE LANGUAGE VERIFICATION

### BACKGROUND OF THE INVENTION

#### Field of the Invention

5 The invention relates to a method to verify preprocessed Java code (Java is a trademark of Sun Microsystems). Essentially it comprises a conversion step after which language verification of non-preprocessed Java code is utilized. In particular, a technical solution is suggested that allows for the conversion of preprocessed Java card CAP files back into standard Java class files. This conversion can be achieved in a provably correct way, thus solving the problem of language verifying Java card CAP files. The proposed concept is also applicable to preprocessed code in languages other than Java.

#### Description of the Related Art

10 Java is an interpreted language defined independently of any particular hardware environment which features high-level language operations. Therefore, Java is extremely well suited for use in embedded microcontrollers. However, due to the size constraints in typical embedded systems, neither the full Java language set, nor the complete execution and loading format as defined for Java on PCs can be sustained in such embedded microcontroller devices. Therefore, it is a common technique to preprocess or convert the regular, PC-oriented Java code, also called class files, into a format suitable for loading and execution on embedded microcontrollers. One concrete example where this has been firmly specified is in the area of the Java card, a chip card featuring a subsetted Java interpreter.

20 One goal of executing Java code is to attain a level of security and trust that can not be achieved by using machine language programs alone. This is due to the interpreted nature of the language and the thus inherent continuous checks of the Java code prior to its actual execution. One common feature providing this trust is a so-called Java class verifier run at development time of the Java code. It checks, for example, static object/class relationships as well as basic scoping rules, i.e., whether a particular method has access to another class's data or code. Programs to perform this work are available.

Java card technology enables programs written in the Java programming language to be run on chipcards, smart cards and other small, resource-constrained devices. Developers can build and test programs using standard software development tools and environments, then convert them into a form that can be installed onto a Java card technology enabled device.

5 Application software for the Java card platform is called an applet, or more specifically, a Java card applet or card applet (to distinguish it from browser applets). While Java card technology enables programs written in the Java programming language to run on chipcards, such small devices are far too underpowered to support the full functionality of the Java platform. Therefore, the Java card platform supports only a carefully chosen, customized subset of the

10 features of the Java platform. This subset provides features that are well-suited for writing programs for small devices and preserves the object-oriented capabilities of the Java programming language. The role of the Java card virtual machine is best understood in the context of the process for production and deployment of Java card software. There are several components that make up a Java card system, including the Java card virtual machine, the Java card converter, a terminal installation tool, and an installation program that runs on the device.

15

Development of a Java card applet begins as with any other Java program. A developer writes one or more Java classes, and compiles the source code with a Java compiler, producing one or more class files. The applet is run, tested and debugged on a workstation using simulation tools to emulate the device environment. Then, when an applet is ready to be

20 downloaded to a device, the class files comprising the applet are converted to a Java card CAP file, also called converted applet file, using a Java card converter. The Java card converter takes as input not only the class files to be converted, but also one or more export files. An export file contains name and link information for the contents of other packages that are imported by the classes being converted. When an applet or library package is converted, the converter can

25 also produce an export file for that package.

After conversion, the Java card CAP file is copied to a card terminal, such as a desktop computer with a card reader peripheral. Then an installation tool on the terminal loads the Java card CAP file and transmits it to the Java card technology enabled device. An installation program on the device receives the contents of the Java card CAP file and prepares the applet

30 to be run by the Java card virtual machine. The virtual machine itself need not load or

manipulate Java card CAP files; it need only execute the applet code found in the Java card CAP file that was loaded onto the device by the installation program. The division of functionality between the Java card virtual machine and the installation program keeps both the virtual machine and the installation program small. The installation program may be implemented as a Java program and executed on top of the Java card virtual machine. Since Java card instructions are denser than typical machine code, this may reduce the size of the installer. The modularity may enable different installers to be used with a single Java card virtual machine implementation.

It would be ideal if programs for chipcards could be written using all of the Java programming language, but a full implementation of the Java virtual machine is far too large to fit on even the most advanced resource-constrained devices available today. A typical resource-constrained device has on the order of 1K of RAM, 16K of non-volatile memory (EEPROM or flash) and 24K of ROM. The code for implementing string manipulation, single and double-precision floating point arithmetic, and thread management would be larger than the ROM space on such a device. Even if it could be made to fit, there would be no space left over for class libraries or application code. RAM resources are also very limited. Therefore Java card technology is implemented as a subset of the Java platform.

Dynamic class loading is for instance not supported in the Java card platform. An implementation of the Java card platform is not able to load classes dynamically. Classes are either masked into the card during manufacturing or downloaded through an installation process after the card has been issued. Programs executing on the card may only refer to classes that already exist on the card, since there is no way to download classes during the normal execution of application code.

Security management in the Java card platform differs significantly from that of the Java platform. In the Java platform, there is a Security Manager class (java.lang.Security Manager) responsible for implementing security features. In the Java card platform, language security policies are implemented by the virtual machine. There is no Security Manager class that makes policy decisions on whether to allow operations.

In order to check at runtime that the Java code has been successfully language verified, typically cryptographic methods are employed. This means that the actual deployer, after having performed a Java language verification, signs the Java code, i.e., the Java card CAP file, in such a way that this cryptographic signature can be undeniably cryptographically verified within the embedded microcontroller system, i.e., a Java virtual machine, that executes the Java code. This way, the embedded microcontroller system does not need to do within its chip the complete Java language verification, which, depending on the hardware capabilities of the embedded microcontroller, would either be a prohibitively expensive or a plainly intractable operation. Therefore, the deployer of the preprocessed Java code establishes the trust in this Java code by means of a Java language verification before he signs the resultant file prior to loading it into the embedded microcontroller system.

The problem faced by the deployers of the preprocessed Java code, e.g., Java card CAP files, is that they no longer have access to the original Java class files to be able to verify those themselves. As it is often, for example for copyright or IP reasons, also not an option to freely provide the Java class files to the deployers of the Java card CAP files, another solution for establishing the trust in the code contained in a Java card CAP file needs to be found. One approach would be to develop a complete verifier operating on preprocessed code, e.g., on Java card CAP files. However, this is a very labor-intensive approach not leveraging the maturity and experiences gained with PC-level Java class verifiers.

## SUMMARY OF THE INVENTION

The invention is directed to a method for language-verification of a Java card CAP file, which has been converted from an original Java code file under conservation of its original Java semantics. In a conversion step the Java card CAP file is converted into a corresponding converted Java code file that is semantically identical to the Java card CAP file. In a language verification step, the converted Java code file is then verified concerning its adherence to Java language specifications. This advantageously provides that a standard tool for language verification can be used, more particularly the verification tool that can be used for language verifying the original Java code file, even more particularly the Java virtual machine. Furthermore the use of a known Java language verification process exhibits the advantage that it has already been tested and corrected with view to programming errors and that the known

process therefore exhibits a much lower number of possible errors than a new process for directly verifying Java card CAP files for their compliance with Java card language rules would have. The invention relies on the experience that has made the language verification of an original Java code file relatively secure and unproblematic.

5       The invention suggests that it is possible to reverse the conversion, i.e., the preprocessing step executed to arrive at a small Java code format, in such a way that it is possible to make use of the existing language verifiers operating on the Java class files. This way, deployers of preprocessed Java code can independently of the developers of the original Java code determine the trust one can put into the original Java program.

10       In accordance with one aspect of the invention there is now provided a method for language verifying Java card CAP files that can be executed on a Java card, i.e., in a device with constrained resources. Constrained resources in this respect means not being able to host the code and processing power that is necessary to provide an on-board language verification. More precisely, the constrained resources do not allow evaluation of the so-called Descriptor Component, specified in the Java card 2.1 VM Specification.

15       Viewing the present invention from another aspect, there is also provided a Java card CAP file language verifier for language verifying Java card CAP files that can be executed on a Java card.

20       The Java card CAP file language verifier comprises a converter for converting the Java card CAP file into a corresponding converted Java code file that is semantically identical to the Java card CAP file, and a language verifier for language verifying the converted Java code file.

25       Although the invention is primarily directed to how language verification can be accomplished for standard Java card CAP files, its scope is not limited to Java card CAP files, but it is also applicable to all scenarios where a reduced file that results from a conversion of an original file is to be language verified. The reduced file differs from the original file in that it results from a reduction operation, also called preprocessing or conversion, and is reduced in size.

      The resulting CAP file contains internal references and the referenced items are named by tokens relative to this CAP file. The CAP file contains references to external libraries as well.

Those external items are also named by tokens relative to packages. The export file contains the mappings of the tokens to the original symbolic names of the corresponding CAP file.

The method according to the invention essentially comprises a conversion step for converting the reduced file into a corresponding converted file that is semantically identical to the reduced file and a language-verification step for verifying the converted file. The language verification verifies whether the converted file is compliant to a predetermined language specification. In case of compliance the result of the language verification is positive and the entity having performed the language verification can create a signature to be attached to the reduced file. This signature, also referred to as second cryptographical signature file, can be cryptographically verified by an end user who can in the case of a positive result thereof execute the corresponding reduced file. Typically, the reduced file is smaller in size than the original file since the medium on which the reduced file is to be delivered is a device with reduced storage and processing capability, also referred to as a device with constrained resources. A mobile device like a chipcard is a typical example of a constrained resource device.

Viewing the present invention from yet another aspect, a language verifier is provided for a reduced file and a method for language verifying a reduced file.

## BRIEF DESCRIPTION OF THE DRAWING

Examples of the invention are depicted in the drawing and described in detail below by way of example. Figure 1 illustrates a schematic depiction of the steps according to the present invention.

## DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following, various exemplary embodiments of the invention are described. First, a description and explanation of the different involved file types is given.

Java programs are represented in compiled, binary form as class files. Java class files are used not only to execute programs on a Java virtual machine, but also to provide type and name information to a Java compiler. In the latter role, a class file is essentially used to document the API of its class to client code. That client code is compiled into its own class file, including symbolic references used to dynamically link to the API class at runtime. Java card technology uses a different strategy for binary representation of programs. Executable binaries and interface binaries are represented in two separate files. These files are respectively called Java card CAP files (for converted applet) and export files.

### The Java class file format

A standard Java class file contains for its class descriptors, namely the class description and the executable instructions for that class in a parseable form. Thus, the Java class file can be read sequentially and converted by a virtual machine at runtime into an executable form. The class description contains for instance information about its super class, its instance and static fields and about its methods and their executable instructions, i.e., the bytecode. Referenced items within the descriptors and within the bytecode are addressed by symbolic names (in UTF8 representation). This includes references to class-internal items as well as references to external items such as classes, fields and methods in external classes for instance in the system library which contains the calls for the operating system on the card. A standard Java class file contains the information for only one class, whereas multiple classes - for instance of one Java package - are typically bundled in a so-called jar file. Thus, the class name

of a class includes the name of its package, i.e., java/lang/object, where java/lang is the package name of the class.

### **The Java card CAP file format**

A Java card CAP file contains a binary representation of a package of classes that can be installed on a device and is used to execute the package's classes on a Java card virtual machine. A Java card CAP file is produced by a Java card converter when a package of classes is converted. A Java card CAP file can contain a user library, or one or more applet definitions. A Java card CAP file consists of a set of components, each of which describes a different aspect of the contents. The set of components in a Java card CAP file can vary, depending on whether the file contains a library or applet definition(s).

The JAR file format is used as the container format for Java card CAP files. What this specification calls a "Java card CAP file" is just a JAR file that contains the required set of Java card CAP components. Java card CAP component files in a JAR file are located in a subdirectory called javacard that is in a directory representing the package. For example, the Java card CAP component files of the package com.sun.framework are located in the directory com/sun/framework/javacard. An export file may also be contained in a JAR file, whether that JAR file contains Java card CAP component files or not. If an export file is included, it must be located in the same directory as the Java card CAP component files for that package would be. Other files, including other Java card CAP files, may also reside in a JAR file that contains Java card CAP component files.

Each Java card CAP file contains all of the classes and interfaces defined in one Java package. Java card converters must be capable of producing Java card CAP files that conform to the specification provided in the Java 2.1. specification. A Java card CAP file consists of a stream of 8-bit bytes. To avoid confusion with the fields of Java card virtual machine classes and class instances, the contents of the structures describing the Java card CAP file format are referred to as items. Unlike the fields of a C structure, successive items are stored in the Java card file sequentially, without padding or alignment.

A Java card CAP file is suited to be run on a Java card and covers a whole Java package and is split into multiple sections. One section, the code section, contains a stripped down



description of all classes of the Java package suitable for the execution on the card and is optimized for a small size and a simple linking scheme on the card. Thus, this section is not intended to be parsed sequentially, but instead is read blockwise and easily adapted for the execution on the card. Other sections on the card, the code description sections, contain information about the classes, fields and methods stored in the Java card CAP file, i.e., it tells where a class description or the bytecode for a method can be found and what the names and types of the individual classes, fields and methods are.

Thus, using the code description section, a Java card CAP file can be parsed and the first part of the original class, method and field information can be built up. However, a Java card CAP file does not contain symbolic references for internally or externally referenced items anymore. Instead, it uses short identifiers, also called tokens, for references which are resolved before execution, for instance:

packages are referenced by an AID, i.e., Application Identifier, 5 to 16 bytes long;

external classes are specified by the package in which they are defined and a short integer which the defining package has reserved for them;

internal classes are specified by their offset within the code section; and

internal and external instance fields are specified by their class and their offset into an instance of this class.

Methods are similarly named, again by using indices, offsets etc. instead of symbolic information. However, symbolic names can be built from the various offsets and indices which do not contain the original symbolic names, but contain equivalent information suitable for a language-verification process, i.e.:

an AID may be converted to a legal Java package name e.g. by creating a symbol starting with "p" and appending the individual bytes of the AID as character;

a legal class name may be formed by taking its converted package name and appending the short identifier as character; and

a legal Java field name may be created by creating a symbolic name starting with "f" and appending the Java card field name, e.g., its offset.

If such a converted Java card CAP file does not contain any references to external items, it can already be language verified by a standard Java virtual machine. If it contains references to external items, the symbolic names referring to them can be mapped to the original symbolic names which can be easily done with the help of Java card export files.

### **The Java card Export File Format**

Export files are not used directly on a device that implements a Java card virtual machine. However, the information in an export file is critical to the operation of the virtual machine on a device. An export file can be produced by a Java card converter when a package is converted. This package's export file can be used later to convert another package that imports classes from the first package. Information in the export file is included in the Java card CAP file of the second package, then is used on the device to link the contents of the second package to items imported from the first package. A Java card export file contains the public interface information for an entire package of classes. This means that an export file only contains information about the public API of a package, and does not include information used to link classes within a package. The name of an export file is the last portion of the package specification followed by the extension '.exp'. For example, the name of the export file of the javacard.framework package must be framework.exp. Operating systems that impose limitations on file name lengths may transform an export file's name according to their own conventions.

A Java card export file describes the mapping of class, field and method names to the identifiers used in the Java card CAP files and is generated by the Java card CAP file generator during the conversion step together with the Java card CAP file itself. These export files are published in case of standardized API's, for instance the system classes. Whenever a Java card application using the system classes is converted, the converter uses this export file to map the symbolic names in the original Java class files to the identifiers described in the export file.

The export files can be used in the other direction as well. Instead of constructing symbolic names for referring to external items, the original names from the export file are

placed in the resulting standard Java class files. The names for only internally visible items are still constructed in the formerly described way.

The so-called Descriptor Component provides sufficient information to parse and verify all elements of the Java card CAP file. It references, and therefore describes, elements in the Constant Pool Component, Class Component, Method Component, and Static Field Component. No components in the Java card CAP file reference the Descriptor Component.

In Figure 1, the scheme starting from an original Java code file 10 and depicting the various stages and files in the process is shown. The original Java code file 10 comprises a class description 11, also labelled CLD, and executable instructions 12, also labelled INS. The original Java code file 10 can be verified for its Java language compliance in an original language verification step E, using a first virtual machine 18. After the language verification, a first signature generator 14 is used to create a first cryptographic signature file 13. The first cryptographic signature file 13 is created only in case of a positive result of the original language verification step E and in that sense confirms the originality and correctness of the original Java code file 10. Thereby, a user, reading and cryptographically verifying the first cryptographic signature file 13 can trust the original Java code file 10, provided the user trusts the issuer of the first cryptographic signature file 13, which is the entity that created the first cryptographic signature file 13. The cryptographical verification can be performed with any known system like the private-public key system or a symmetric key system, whereby the key is available to the user, typically on the card 25.

For making the original Java code file 10 executable on a mobile medium with constrained resources, such as a Java card 25, the original Java code file 10 is converted in an original conversion step D into a Java card CAP file 20 which has reduced size with respect to the original Java code file 10. This conversion is done with a Java card CAP file generator 15. The resulting Java card CAP file 20 comprises a code section 21, also labelled COD, and a code description section 22, also labelled CDE. A further result of the original conversion step D is a Java export file 23, also labelled EXP, which contains essentially the information which is necessary to convert the Java card CAP file 20 back into its original class file 10, whereby this back-conversion need not lead to the exact original class file 10 but to a semantically identical file. In the original conversion step D the Java card CAP file contents, i.e., the code

and code description sections 21, 22, are created in such a way that the original Java semantics are conserved, e.g., changes, to the bytecodes and descriptors occur in such a way that the standard language verification rules still apply. This is for instance true for the Java card 2.1 standard.

Also available together with the Java card CAP file 20 may be the first cryptographic signature file 13. The recipient of the Java card CAP file 20 may then simply put the Java card CAP file 20 together with its first cryptographic signature file 13 on the card 25. However there are several reasons why this might not be possible or recommended. Since the first cryptographic signature file 13 only establishes trust in the original Java class file 10, the user of the Java card CAP file 20 on the Java card 25 can not automatically trust the Java card CAP file 20. The original conversion step D could have been performed by some untrusted entity delivering as the Java card CAP file 20 some incorrect code. Also, a signature does per se not tell the user what tests the signing entity has performed. A language verification has not necessarily taken place. Hence, the user of the Java card CAP file 20 is only safe when he can establish trust in the Java card CAP file 20 itself. Therefore this Java card CAP file 20 should be verified itself. Also a typical situation could be that the user has the Java card CAP file 20 but not the original Java class file 10. He might also not even have the first cryptographic signature file 13. The user wants to establish trust in the Java card CAP file 20 and therefore needs a language-verification process. As user in this context is to be seen an entity that has the Java card CAP file 20 and wants to load it to the card 25. This loading step shall be done including a signature that can be cryptographically verified and that to an end user of the card 25 signals that the Java card CAP file 20 has been Java-language verified.

The Java card 25 has no space for hosting an on-card verifying tool like the Java card Descriptor Component. So a direct on-card language verification of the Java card CAP file 20 is not possible.

### **Verification of the Java card CAP file**

A language verification of the Java card CAP file 20 is made possible by the following method.

In a conversion step A, the Java card CAP file 20 is converted into a converted Java code file 40 which is semantically identical to the Java card CAP file 20. Since the Java card CAP file 20 is itself semantically identical to the original Java class file 10, the original Java class file 10 and the converted Java code file 40 are semantically identical as well. Although during any of the conversion steps D, A some information gets lost, this is not relevant for the language-verification steps E, B.

Afterwards, a language-verification step B follows which can make use of a standard virtual machine as second virtual machine 19 for language verification. So the second virtual machine 19 is a verification virtual machine VVM, just as the first virtual machine 18 is also a verification virtual machine VVM. Again, only in case the language-verification step B leads to a positive result, a cryptographic signature file, here a second cryptographic signature file 43 is created in a signature step F and can be used to establish trust in the Java card CAP file 20. For creating the second cryptographic signature file 43, a second signature generator 44 is used. Ideally, the second virtual machine 19 is identical in its functionality to the first virtual machine 18. The second virtual machine 19 according to the present technology can not be installed on the Java card 25 itself, so the Java-language verification is performed off card. The second cryptographic signature file 43 can then be attached to the Java card CAP file 20. During a loading step for loading the Java card CAP file 20 to the Java card 25, the second cryptographic signature file 43 can be used in that a signature is calculated for the Java card CAP file 20 and compared with the second cryptographic signature file 43. In case there is identity between the second cryptographic signature file 43 and the calculated signature, the Java card CAP file 20 can be stored on the Java card 25. The second cryptographic signature file 43 then can but need not be stored on the Java card 25. On the Java card 25 there is another virtual machine installed which is an executing virtual machine 24, labelled EVM and which can execute the Java card CAP file 20.

The conversion step A can be split into two steps, a preconversion step A1 and a mapping step A2. The preconversion step A1 serves for converting Java card IDs contained in the Java card CAP file 20 into symbolic names, and for converting the Java card CAP file 20 into a standard Java format.

Optionally for the conversion step A, the Java export file 23 can be used. The internal references in the Java card CAP file 20 which are expressed in form of tokens can then be combined with the symbolic information stored in the Java export file 23. Internal items can then be named by the tokens and the original symbolic names. This is better readable but in principle not necessary to obtain semantical identity.

A preconverted file 30 is thereby obtained. The preconverted file 30 itself contains a preconverted code section 31, also labelled COD', and a preconverted code description section 32, also labelled CDE'.

The mapping step A2, performed by a mapper 17, serves for replacing in the preconverted file 30 externally defined names with original names by using a mapping scheme between Java names and tokenized identifiers, short tokens, to obtain the converted Java code file 40 for the language-verification step B. This step is only necessary if such externally defined names are actually present in the preconverted file 30. The converted Java code file 40 contains a converted class description 41, also labelled CLD', and a section with converted executable instructions 42, also labelled INS'.

The mapping step A2 is performed under use of a referenced Java export file 26 which is available as a result from a conversion step in which a referenced Java card CAP file has been converted from its original Java code file. Whereas in the conversion step A only internal references have been resolved, in the mapping step external references are resolved. Such a reference can be directed to a different class file than the original class file 10. It is assumed that for that different class file, herein referred to as referenced class file, a corresponding Java card CAP file, referred to as referenced Java card CAP file, and a corresponding export file, referred to as referenced export file 26, are available. Hence for resolving a reference to such a referenced class file, present in the preconverted file 30, the corresponding referenced export file 26 is used in the mapping step A2. Thereby the external references are resolved and converted into symbolic external references. This is done with every referenced Java export file 26 of a referenced class file that is referenced by the Java card CAP file 20. All the external items, referenced in the Java card CAP file 20, which are named by tokens so far, are hence associated with their original symbolic names.

The converted Java code file 40 hence has all references converted into symbolic references.

Finally the preconverted class file 30 is converted into a standard class file by rearranging the individual code section, header section and link information so that they comply with the standard class file format. This is possible since all the former transformations from the original Java class file 10 to the Java card CAP file 20 did not change the semantics, but only the symbolic information has been replaced by tokens and the layout has been changed.

For the language-verification step B, the second virtual machine 19 accesses the referenced class files in order to perform the language verification. For each of the referenced class files two possibilities exist. First, they are system class files and hence directly accessible by the second virtual machine 19, or second they are non-system class files, also referred to as libraries. Those libraries are typically again accessible in form of their corresponding CAP files, also referred to as library CAP files. For those library CAP files, the proposed method of conversion is used recursively, i.e., the corresponding converted Java code file for that library CAP file is generated and checked for references to other libraries. This is continued until the second virtual machine 19 arrives at a situation when all external references are to system classes only. Then the second virtual machine 19 has all information needed for language-verifying the converted Java code file 40.

The language-verification step B leads to the creation of the second cryptographic signature file 43 when its result is positive. The Java card CAP file 20 is joined with the second cryptographic signature file 43 and loaded to the card 25. An end user can now read the second cryptographic signature file 43 and cryptographically verify it. The Java card CAP file 20 can be executed via the executing virtual machine 24, once trust has been established via the language-verification step B.

With the above described method, the problem of Java-language verifying the Java card CAP file 20 is reduced to the language verification of its underlying Java class file 10, respectively the semantically identical converted Java code file 40. This is easier than a direct Java-card language verification of the Java card CAP file 20 since the language-verification step B can make use of the same facility as the original language-verification step E has done.

The virtual machine to be used therefor is already a well known and reliable tool. By this method the development of a direct Java-card language verifier operating on the Java card CAP file 20 is rendered obsolete.

In case of the Java card 25, hence the executable content which is stored in the standardized Java card format is converted into a file semantically identical to the Java class file 20 in the standard Java class file format which is used for the language verification by a standard Java virtual machine, i.e., the second virtual machine 19.

### Detailed Example

In the following, the original conversion step D and the conversion step A are shown for a part of the class information. Here, only considered is the information in a class regarding its name, the name of its super class and its fields. The remaining information for methods, etc., is however processed in an analogous manner.

The original Java code file 10 in the standard class file format contains the following information regarding name, super class name and fields:

the name is given in UTF8, e.g. symbolic;

the name of the super class is given symbolic (UTF8); and

a field contains its name (symbolic), type (symbolic), access rights (PUBLIC, PRIVATE, etc.).

During the conversion process to the Java card CAP file 20 which here is a Java card 2.1 CAP file, the following process is carried out. The symbolic names are mapped to short identifiers and the information is split into three parts where two parts, the code description section 22 and the code section 21, are included in the Java card 2.1 file, and the third part is separately stored as the Java export file 23. The export file 23 contains the symbolic information for the class, its super class and the fields. It also contains the identifiers which are used for these names, classes and fields.

The Java card CAP file 20 is basically split into two parts:



1. An execution component, which is the code section 21, with minimal class information, for instance the super class identifier) and the instructions; and
2. A description component, which is the code description section 22 contains the information where the individual classes can be found, the fields they have, etc. For instance, a class is specified by giving its identifier and an offset into the execution component where additional information can be found.

So the information can be gathered to reconstruct standard class files, i.e., to convert the Java card CAP file 20 into the preconverted file 30. From the execution component 21 and the description component 22, one can get the total class information with the limitation that only short identifiers for the identification of fields, classes and methods are available. From the Java export file 23, one can get the symbolic information for the identifiers. Then a standard class file, as is the converted Java code file 40, can be constructed.

The generated standard class file, i.e., the converted Java code file 40, is not the same as the original class file 10, but it is semantically equal to it. The Java export file 23 contains only the symbolic names for public or protected elements, not for private elements. That means that in the original Java code file 10 the name of a field could have been "MARCUS", but that information is not available after the original conversion step D anymore. Instead, the Java card CAP file 20 will declare that field, but with a short identifier like "2". The regenerated standard class file, i.e., the converted Java code file 40, will then declare the field name to be, for example, "f2". It will have the same type (integer, byte etc.). Thus, at execution time the regenerated standard class file, i.e., the converted Java code file 40, will behave exactly the same as the original Java code file 10. Method names and class names might have to be constructed like that as well, but again, no difference at execution time can be recognized.

The Java card CAP file 20, i.e., the executable component which is the code section 21 and the descriptor component, which is the code description section 22, contains the information about its methods and their instructions. In theory the conversion step A could optimize the Java code in such a way that the code in the regenerated standard class file 40 is not a valid Java bytecode anymore and would be rejected by the standard Java language verifier 19. However, Java card CAP files 20 must be verifiable themselves and the rules applying for

Java card 2.1 bytecode are the same as for standard Java. That also means that all Java card 2.1 instruction conversions do not affect the properties as they are defined by standard Java.

### Generalization

Although the invention has been described in detail for the Java language, it is applicable to other environments and languages which may be manipulated in a similar manner. A generic description of the process follows.

An original file 10 containing a description section 11 and an instruction section 12, written in a condensed interpreted language is converted into a reduced file 20, comprising a code section 21 and a code description section 22, while conserving of its language semantics. This reduced file is in the hands of an entity which does not have direct access to the original file 10 but wants to verify the reduced file 20 with respect to its compliance with the rules of the condensed interpreted language. For this language verification, the reduced file 20 is converted into a converted file 40, comprising a converted description section 41 and a converted instruction section 42. The converted file 40 is semantically identical to the original file 10, respectively to the reduced file 20. A loss of information during that step is accepted which does not influence the correctness and authenticity of the respective code and hence does not influence the result of the language verification itself. Finally, the converted file 40 is language verified. This can advantageously be done with the same or a similar tool with which a language verification of the original file 10 can be performed. In the case of a positive language verification result, the entity can create a second signature file 43 with which the reduced file 20 is signed. The reduced file 20 may then be loaded together with the second signature file 43 to a data carrier. This data carrier, also called storage device, can even be an environment 25 with constraint resources, which means that the reduced file 20 is not directly verifiable in that environment 25, e.g., a mobile device like a chipcard. The environment 25 can be taken by an end user who can cryptographically verify the second signature file 43 and upon a positive result thereof, provided there is trust in the entity that created the second signature file 43, trust the reduced file 20 and finally have it executed by the executing virtual machine.

The conversion step A for arriving from the reduced file 20 at the converted file 40 can be split into two substeps, namely the preconversion step A1 and the mapping step A2. The

[illegible]

5

15

20

25

which can be converted. This is recursively repeated until finally only available system files or original files are referenced. Then the language-verification step B has access to all information needed. In case of a positive verification result, a signature file 43 can be created and attached to the reduced file 20.

5 It is obvious to the person skilled in the art that the present invention can be realized in hardware, software, or a combination of these. Also, it can be implemented in a centralized fashion on one single computer system, or in a distributed fashion where different elements are spread across several interconnected computers or computer systems, whereby any kind of a computer system, or other apparatus adapted for carrying out the methods described herein, is suited. A typical combination of hardware and software could be a general-purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein. The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described herein, and which, when loaded in a computer system, is able to carry out these methods.

10 Computer program means or computer program in the present context mean any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of conversion to another language, code or notation and/or reproduction in a different material form.

20 Any disclosed embodiment may be combined with one or several of the other embodiments shown and/or described. This is also possible for one or more features of the embodiments. It is obvious that a person skilled in the art can modify the shown arrangements in many ways without departing from the gist of the invention which is encompassed by the subsequent claims.

25

arrangements in many ways without departing from the gist of the invention which is encompassed by the subsequent claims.

11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
2213  
2214  
2215  
2216  
2217  
2218  
2219  
2220  
2221  
2222  
2223  
2224  
2225  
2226  
2227  
2228